Exhibit F

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner,

v.

PARTEC CLUSTER COMPETENCE CENTER GMBH,
Patent Owner.

IPR2025-00318
U.S. Patent No. 11,537,442
Issued: December 27, 2022
Application No. 16/963,749
Filed: January 23, 2019

Title: Application Runtime Determined Dynamical Allocation Of Heterogeneous Compute Resources

PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 11,537,442

TABLE OF CONTENTS

TABLE OF AUTHORITIES

LIST OF EXHIBITS

No.	Description
1001	U.S. Patent No. 11,537,442 ("**'442 patent**")
1002	File History of U.S. Patent No. 11,537,442
1003	Declaration of Darrell Long, dated December 20, 2024 ("**Long Decl.**" or "**Long**")
1004	U.S. Patent App. Pub. No. U.S. 2013/0282787 ("**Lippert**")
1005	John R. Budenske et al., *A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms*, The Journal of Supercomputing, Vol. 12, No. 4 (Oct. 1998) ("**Budenske**")
1006	U.S. Patent App. Pub. No. U.S. 2018/0074855 ("**Kambatla**")
1007	Yu-Kwong Kwok et al., *A Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems*, Journal of Parallel and Distributed Computing (2005) ("**Kwok**")
1008	Mitchell D. Theys et al., *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach*, Solutions to Parallel and Distributed Computing Problems (A. Zomaya, F. Ercal & S. Olariu eds., 2001) ("**Theys**")
1009	U.S. Patent No. 3,496,551 ("**Driscoll**")
1010	Yuan-Chieh Chow and Walter H. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System*, IEEE Transactions on Computers, Vol. C-28, No. 5 (May 1979) ("**Chow**")
1011	David M. Nicol and Paul F. Reynolds, Jr., *Optimal Dynamic Remapping of Data Parallel Computations*, IEEE Transactions on Computers, Vol. 39, No. 2 (Feb. 1990) ("**Nicol**")
1012	Arun C. Murthy, et al., Apache Hadoop YARN (2014) ("**Murthy**")
1013	Morris Jette and Mark Grondona, *SLURM: Simple Linux Utility for Resource Management*, ClusterWorld Conference and Expo ("**Jette**")

No.	Description
1014	Norbert Eicker et al., *The DEEP Project: An Alternative Approach to Heterogeneous Cluster-Computing in the Many-Core Era*, Concurrency and Computation: Practice and Experience (2016) ("**Eicker**")
1015	U.S. Patent App. Pub. No. U.S. 2017/0262319 ("**Newburn**")
1016	Carsten Clauss et al., *Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing*, Proceedings of the First Workshop on Co-Scheduling of HPC Applications (2016) ("**Clauss**")
1017	Declaration of Rachel J. Watters dated December 13, 2024 ("**Watters Decl.**")
1018	Samik Raychaudhuri, *Introduction to Monte Carlo Simulation*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Raychaudhuri**")
1019	David W. Bauer, Jr. and Mojdeh Mohtashemi, *An Application of Parallel Monte Carlo Modeling for Real-Time Disease Surveillance*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Bauer**")
1020	List of Challenged Claims
1021	U.S. Patent No. 5,881,227 ("**Brenner**")
1022	U.S. Patent No. 7,047,299 ("**Curtis**")
1023	N. Metropolis, *The Beginning of the Monte Carlo Method*, Los Alamos Science, Special Issue (1987) ("**Metropolis**")
1024	U.S. District Court, Eastern District of Texas, Calendar Events Set for 4/1/2026-4/30/2026, Judge Robert W. Schroeder, III, Presiding
1025	Time to Milestones Report, Judge Robert W. Schroeder, III, DocketNavigator, https://search.docketnavigator.com/patent/judge/15325/15?print=true (last visited Dec. 19, 2024)

MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

1. **Real Party-In-Interest**

Microsoft Corporation is the sole real party-in-interest.

2. **Related Matters**

The ’442 patent (EX1001) has been asserted in the following litigations:

- *ParTec AG et al. v. Microsoft Corp.*, Case No. 2:24-cv-00433 (E.D. Tex.), filed June 10, 2024.

3. **Lead And Back-Up Counsel, And Service Information**

Lead Counsel	**Back-up Counsel**
Andrew M. Mason (Reg. No. 64,034) andrew.mason@klarquist.com	Cameron D. Clawson, Reg. No. 73,509 cameron.clawson@klarquist.com Sarah Jelsema, Reg. No. 70,804 sarah.jelsema@klarquist.com Frank Morton-Park, Reg. No. 80,750 frank.morton-park@klarquist.com
KLARQUIST SPARKMAN, LLP 121 SW Salmon Street, Suite 1600 Portland, Oregon, 97204 503-595-5300 (phone) 503-595-5301 (fax)	

Petitioner consents to service via email at the above email addresses and the email address of Msft-Partec@klarquist.com.

Pursuant to 37 C.F.R. § 42.10(b), concurrently filed with this Petition is a Power of Attorney executed by Petitioner and appointing the above counsel.

Petitioner authorizes Account No. 02-4550 to be charged for any fees, including those enumerated in 37 C.F.R. § 42.15.

I. INTRODUCTION

Microsoft Corporation ("Petitioner") respectfully requests *inter partes* review ("IPR") of claims 1-10 of U.S. Patent No. 11,537,442 ("the '442 patent") (EX1001), allegedly assigned to ParTec Cluster Competence Center GmbH ("Patent Owner"). For the reasons set forth below, these claims should be found unpatentable and cancelled.

The challenged claims of the '442 patent relate to distributing sub-tasks of a computation task among processors of a heterogeneous computing system and adjusting the distribution for subsequent iterations of the computation task. *E.g.*, EX1001, Claim 1. As distributing sub-tasks among processors of a heterogeneous computing system was already known, the alleged point of novelty of the challenged claims was processing sub-tasks in "multiple computing iterations" and redistributing sub-tasks "based on information relating to the processing of the plurality of sub-tasks." EX1002, 174-75.

Twenty years before the filing date of the '442 patent, however, Budenske (EX1005) disclosed the same concept of processing an iterative application with a heterogeneous computing system and dynamically re-mapping sub-tasks between iterations based on "dynamic parameters of the application" relating to processing of the sub-tasks in a preceding iteration. EX1005, 391. These teachings by Budenske, when considered with more modern teachings on heterogeneous computing systems as in Lippert (EX1004) and Kambatla (EX1006), render each challenged claim obvious.

II. GROUNDS FOR STANDING PER 37 C.F.R. § 42.104(a)

Petitioner certifies that the '442 patent is available for IPR and that Petitioner is not barred or estopped from requesting an IPR challenging the patent claims on the grounds identified in this petition.

III. IDENTIFICATION OF CHALLENGE

A. Statement Of The Precise Relief Requested / Statutory Grounds

Petitioner requests *inter partes* review of claims 1-10 (the "Challenged Claims") of the '442 patent, on the following statutory grounds:

	Reference(s)	**Basis**	**Claims**
Ground 1	Lippert (EX1004) and Budenske (EX1005)	35 U.S.C. § 103	1-10
Ground 2	Lippert, Budenske, and Kambatla (EX1006)	35 U.S.C. § 103	2-5, 8-10

For each ground, in Sections VII and VIII below, the petition presents evidence of unpatentability and establishes a reasonable likelihood that the Petitioner will prevail in establishing that each Challenged Claim is unpatentable.

B. No Examiner Addressed These Unpatentability Grounds

Prior art references Budenske (EX1005) and Kambatla (EX1006) were neither cited nor discussed during prosecution and were not referenced in the '442 patent.

Budenske provides express teachings on remapping iterative tasks onto heterogeneous parallel hardware platforms between iterations. Kambatla provides express teachings on cluster resource management. These teachings, which show that the challenged claims are unpatentable as obvious, were not considered by the Examiner.

Prior art reference Lippert (EX1004) was considered during prosecution of the '442 patent, with the Examiner rejecting all claims for being anticipated by Lippert. EX1002, 156-66.[1] Nothing in the record, however, indicates that the Examiner considered a combination of Lippert with Budenske, or of Lippert with Budenske and Kambatla, much less one informed by expert testimony, as relied on herein. By failing to recognize the long-known technique of re-distributing sub-tasks among resources of a heterogeneous computing system between iterations, as taught by Budenske, the Examiner materially erred in allowing the challenged claims.

Because these key teachings and combinations of the prior art were not previously considered by the Office, this Petition should not be denied under Section 325(d).

[1] EX1002 file history citations are to page numbers added by Petitioner.

C. *Fintiv* Denial Is Improper

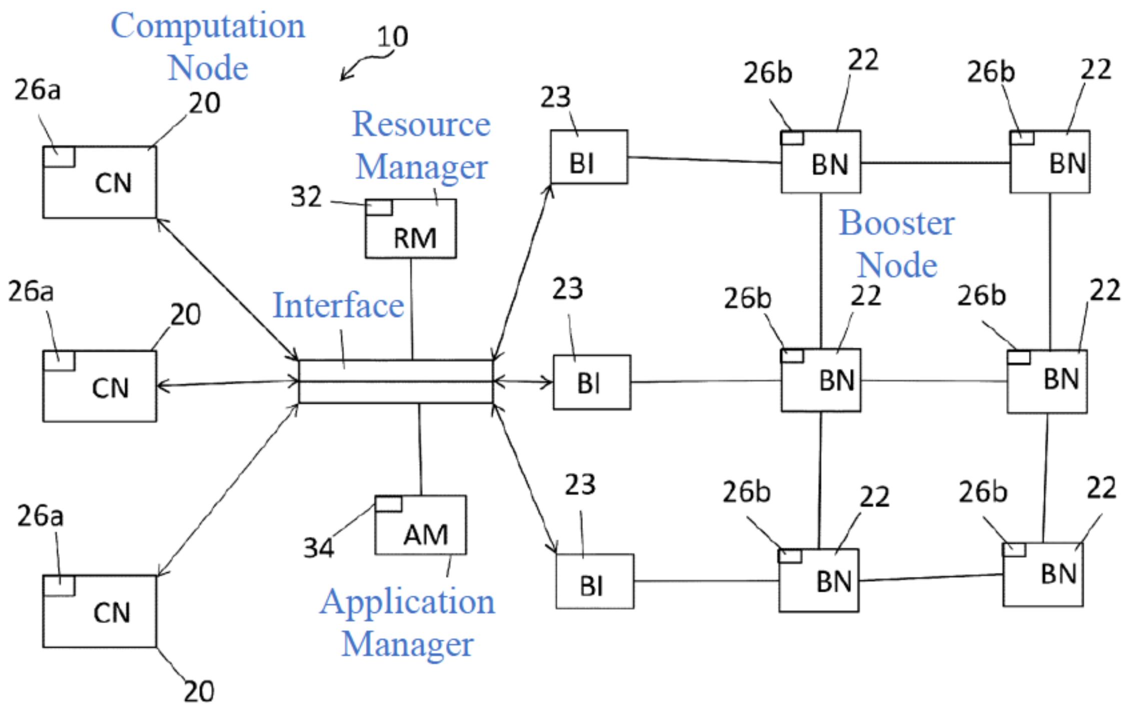
The Board should not discretionarily deny institution under *Fintiv* because Petitioner promptly filed this Petition less than four months after receiving infringement contentions on September 4, and less than two months after serving invalidity contentions on October 30. The district court case is in its early stages, before any ESI requests or production and before any depositions have been taken. While the court set a jury trial for April 20, 2026, roughly eight weeks before the expected final written decision (FWD) deadline, any actual jury trial seems likely to occur much later because (1) the district court judge has set four other cases for jury trial on April 20, 2026 (EX1024, 1-3), and (2) the district court judge's median time to jury trial is 27.0 months (EX1025, 2), which would put trial in September 2026. Finally, the Petition presents compelling merits favoring institution, as prior art Budenske was not previously considered and squarely teaches the alleged point of novelty. *Infra* Section VII.

IV. THE '442 PATENT

The '442 patent issued December 27, 2022, from a U.S. national stage application of International Patent Application No. PCT/EP2019/051615 filed January 23, 2019, and alleging priority to European Patent Application No. EP18152903.3 filed January 23, 2018. EX1001, Cover, 1:8-12.

A. The '442 Patent's Specification

The '442 patent describes a "cluster computing system 10" comprising "a number of computation nodes 20 and a number of booster nodes 22" that are "arranged to compute a computation task … comprising a plurality of sub-tasks." EX1001, 1:66-2:5, 2:66-3:2. A computation node includes, e.g., a "multi-core processor," while a booster node includes, e.g., an "accelerator-type processor." *Id.*, 3:25-27, 3:33-36. The '442 patent explains that these devices and architecture were known in the art, with booster nodes known and used for processing "highly scalable code parts" or sub-tasks of a computation task. *Id.*, 1:23-55, 3:25-38, 3:53-55.



EX1001, FIG. 1 (annotated); EX1003, ¶43.

The cluster includes a resource manager and an application manager, as well as daemons associated with each element of the cluster where each daemon "runs as a background process" in the respective element. EX1001, 3:6-23. The application manager collects information from daemons at the nodes and provides this information to the resource manager "to determine if the distribution of sub-tasks between computation nodes and booster nodes can be adapted or improved upon for a further computing iteration." *Id.*, 2:13-25; EX1003, ¶43.

According to the '442 patent, "[a] job to be computed by the system may comprise a number of tasks some of which or all may be repeated a number of times during the execution of the job." EX1001, 3:44-49. The '442 patent briefly describes distributing sub-tasks between computation nodes and boosters, but that an initial distribution of sub-tasks between a computation node and a booster during a first iteration may not be optimal, so altering the distribution of the sub-tasks among nodes for later iterations "might improve the efficiency of the computation of the task." *Id.*, 3:59-4:3; EX1003, ¶44.

B. <u>The Prosecution History</u>

The first Office action interpreted the elements "a plurality of computation nodes, a plurality of booster nodes, communication interface, resource manager, and application manager" in presented claim 9 under Section 112(f), and rejected all presented claims 1-10 as anticipated by U.S. 2012/0317168, Driesen et al. EX1002, 123-37. The Examiner then acknowledged during an interview that the incorrect prior art reference was cited in the first Office action, and informed the Applicant that Lippert was the correct prior art reference. *Id.*, 151-52.

The second Office action maintained the interpretation of the elements of claim 9 under Section 112(f) and rejected all claims as anticipated by Lippert. EX1002, 155-66. In response, Applicant argued that the specification includes sufficient structural support for the claim limitations to avoid Section 112(f) interpretation (*id.*, 173), and that Lippert does not anticipate the claims because Lippert does not disclose processing the plurality of sub-tasks in "multiple computing iterations." *Id.*, 174 (emphasis omitted). Applicant further argued that the claims were distinguished from Lippert because they "allow for redistribution based on information relating to the processing of the plurality of sub-tasks," whereas according to Applicant, "Lippert merely proposes a load balancing based on the workload of the nodes." *Id.*, 174-75; EX1003, ¶¶45-47.

The Examiner then allowed all pending claims. EX1002, 184.

C. <u>The Challenged Claims</u>

Challenged claim 1 is representative and listed here, with element numbering added in the lefthand column:

[1.1]	A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being

	arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:
[1.2]	in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and
[1.3]	generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

A full claims listing is submitted as Exhibit 1020.

V. LEVEL OF SKILL IN THE ART

A. Person Of Ordinary Skill In The Art

The person of ordinary skill in the art in January 2018 ("POSITA") would have held a bachelor's degree in computer science, electrical engineering, computer engineering, or a closely related field, and approximately three years of experience working with distributed computing technologies. More education, e.g., a Master's or Ph.D., would compensate for less work experience and vice versa. EX1003, ¶¶29-31. Such education and industry experience working with distributed computing technologies would have included designing, implementing, and maintaining distributed computing systems, such as cluster computing systems, and deploying applications on such systems. The state of the art identified below reflects some of the knowledge and skill of a POSITA. EX1003, ¶¶32-42.

B. State Of The Art

POSITAs have long known how to assign tasks between multiple processors in heterogeneous computing systems. *E.g.*, EX1009, 1:70-2:4; EX1010, 354; EX1003, ¶25. A well-known "design goal of any multi-processor computing system is to achieve high overall efficiency." EX1009, 2:13-20. To achieve overall efficiency throughout processing of a task, POSITAs have long known that tasks should be assigned to processors *dynamically* because "[s]ome computations dynamically change their behavior as they progress." EX1011, 206; EX1003, ¶26.

One known type of distributed computing architecture is a "cluster," which is a collection of interconnected computing devices ("nodes") utilized as a single, unified computing resource. EX1022, 1:16-54. Clusters often use "daemons," a daemon being a background process that performs a specific task. EX1021, 7:66-67, 8:2-3, 2:54-3:18; EX1022, 3:63-4:14. As of the earliest alleged filing date, POSITAs typically used cluster resource management frameworks such as Hadoop YARN or SLURM to manage allocation of jobs to cluster resources. EX1012, xvii-xviii, 1-20, 43-58; EX1013, 3-4. Such frameworks may include a daemon at every cluster node to enable monitoring and control of the node by a central resource management daemon. EX1012, 9, 38-40; EX1013, 5-7, 20; EX1003, ¶27.

The '442 patent includes various admissions regarding the state of the art. For example, the '442 patent admits that heterogeneous computing systems comprising a plurality of computation nodes and a plurality of booster nodes were known. EX1001, 1:23-27. The '442 patent admits that prior art reference Eicker (EX1014), describes such a heterogeneous computing system capable of "taskifying" an application into a plurality of tasks that are distributed among the nodes to enable scalability. *Id.*, 1:39-55. The '442 patent admits that Newburn (EX1015) describes "runtime processes that can fully or partially automate the distribution of data and mapping of tasks to computing resources." *Id.*, 1:56-58; *see also id.*, 5:10-24. The '442 patent further admits that resource managers for "dynamic process management" were known and "described in Clauss" (EX1016), a prior art reference which also discloses "daemons as addressed" in the '442 patent. *Id.*, 1:28-35, 3:13-17. As an example of a job with tasks that may be repeated during execution, the '442 patent suggests a "'Monte-Carlo' based simulation," a well-known method in the art that was also known for efficient processing with cluster computing systems. *Id.*, 3:44-49; EX1018, 91; EX1019, 1029-36; EX1023, 125-130; EX1003, ¶28.

VI. CLAIM CONSTRUCTION

Unless otherwise expressly discussed below, Petitioner applies the plain and ordinary meaning of all claim terms, without waiving any rights to raise additional issues of claim construction, indefiniteness, or invalidity in any litigation. EX1003, ¶48.

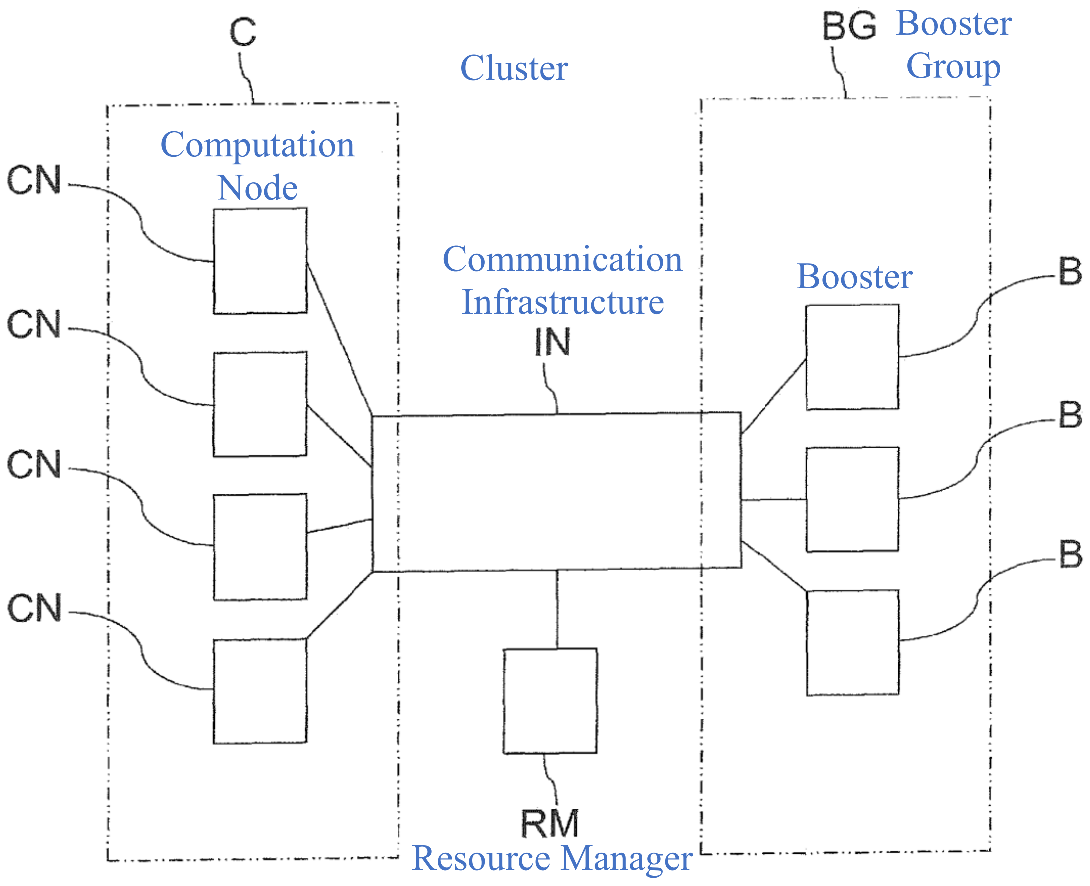
VII. GROUND 1: CLAIMS 1-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE

As explained below, claims 1-10 are obvious over Lippert in view of Budenske.

A. Lippert (EX1004)

Lippert is a U.S. patent application published on October 24, 2013, and thus qualifies as prior art under at least Section 102(a)(1). EX1004, Cover. Lippert is analogous art to the ’442 patent partly because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1004, [0002]; EX1003, ¶¶49-50.

Lippert describes a “computer cluster arrangement for processing a computation task,” the arrangement comprising a plurality of computation nodes and a plurality of boosters. EX1004, [0009]-[0011], FIG. 2. In Figure 2, Lippert depicts an example computer cluster with a plurality of computation nodes CN and a plurality of boosters B in a booster group BG, the computation nodes CN and the boosters B are interconnected via a communication infrastructure IN. *Id.*, [0067]. To process a computation task, a resource manager RM “initializes a first assignment and further on establishes a dynamic assignment of boosters B to computation nodes CN.” *Id.*, [0068], [0013].



EX1004, FIG. 2 (annotated); EX1003, ¶51.

The computation task is allocated among computation nodes and boosters in part based on the processing requirements of each part of the computation task and the processing capacities of processing resources, where computation nodes compute a first part of the computation task while a booster computes a second part of the computation task. EX1004, [0017], [0010]-[0012], [0014]-[0016], [0018]-[0021], [0027]-[0029]; EX1003, ¶52.

B. Budenske (EX1005)

Budenske is a printed publication published October 1998 in Volume 12, Number 4 of the Journal of Supercomputing. EX1005, 387. Budenske bears a 1998 copyright and an ISSN of 0920-8542, indicia of publication. *Id.*, ii-iv. Budenske further indicates that the Journal of Supercomputing was published by Kluwer Academic Publishers, a well-known and reputable publisher, and abstracted and/or indexed in a variety of citation databases. EX1005, iii-iv. Budenske has been sufficiently accessible to the public interested in the art since its publication, and thus qualifies as prior art under Section 102(a)(1). EX1017, 1-3 (librarian attesting that Budenske was available to the public in 1998). Budenske is analogous art to the '442 patent partly because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1005, 387; EX1003, ¶53.

Budenske teaches the execution of iterative tasks with a "heterogeneous parallel hardware platform," and the real-time re-mapping of sub-tasks to processors between iterations based on the sub-tasks' processing of input data in a preceding iteration. EX1005, 387. An "Intelligent Operating System" (IOS) "make[s] a heuristically-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data." *Id.*, 387-88. A central module of the IOS (HC Kernel or HC Kernel

Monitor) (1) establishes an initial mapping of the sub-tasks onto the processing resources of the heterogeneous computing system, (2) monitors the execution of the sub-tasks, and (3) at the end of each iteration, decides how to change the mapping of the sub-tasks onto the hardware based on information about dynamic parameters obtained from processing the sub-tasks in the preceding iteration. *Id.*, 401-02; EX1003, ¶54.

Budenske teaches determining a new mapping of sub-tasks to processing resources "in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration." EX1005, 402. Budenske teaches "reconfigur[ing] the assignment of processing resources to subtasks **to reduce execution time** of the next iteration." *Id.*, 391 (emphasis added); EX1003, ¶55.

C. Combining Lippert And Budenske

A POSITA would have been motivated, with a reasonable expectation of success, to combine Lippert's teachings on a computer cluster arrangement with Budenske's teachings on re-mapping sub-tasks between iterations of an iterative application. A skilled artisan would have combined these respective teachings into an improved design (hereinafter "Lippert-Budenske") that included benefits from each reference. EX1003, ¶56.

The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
  - A plurality of computation nodes;
  - A plurality of booster nodes; and
  - A communication infrastructure interfacing each of the computation nodes and the booster nodes;
- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02); and
  - Dynamically assign, during processing of the computation task, computation nodes and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);
- An application manager module, as taught by Budenske (*e.g.*, EX1005, 388, 402), arranged to:

- Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);
- Monitor execution of sub-tasks to collect information relating to processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
- Assign, according to a new mapping obtained using the information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02).

EX1003, ¶57.

A POSITA would have been motivated, with a reasonable expectation of success, to so combine the teachings of Lippert and Budenske for at least the following reasons. EX1003, ¶58.

First, Lippert and Budenske each discloses heterogeneous computing systems that process an application or computation task with dynamic assignments. *E.g.*, EX1004, [0015]; EX1005, 387. Lippert discloses a flexible assignment of boosters to computation nodes to facilitate scalability, where parts of a computation task are allocated between computation nodes and boosters. *E.g.*, EX1004, [0013], [0015]-

[0020]. Budenske discloses a flexible assignment of sub-tasks of an iterative application to processors of a heterogeneous computing system, so that each sub-task is assigned "where it is best suited for execution" for each iteration. EX1005, 390, 387. A POSITA would have been motivated to implement the heterogeneous computing system of Lippert with the dynamic iterative sub-task assignment method of Budenske partly because doing so would facilitate efficient distributions of sub-tasks and processing resources during execution of an iterative application, thus fully leveraging the flexibility of the heterogeneous architecture and "reduc[ing] execution time" of computing iterations. EX1005, 391; EX1003, ¶59.

Second, Budenske provides express motivation for implementing its method in a heterogeneous computing system such as Lippert, namely to "fully utilize the architectural flexibility of such a [heterogeneous parallel computing] system." EX1005, 390. A POSITA would have found that the dynamic flexibility of the heterogeneous computing system taught by Lippert through the "loose coupling" of computation nodes and boosters (EX1004, [0013], [0015] (describing "a maximum flexibility in providing hardware resources")), naturally complements the dynamic flexibility of remapping sub-tasks among processing resources to account for the actual processing of the sub-tasks by the processing resources during preceding iterations, as taught by Budenske (EX1005, 391), and that the combination of these teachings would provide increased flexibility for optimally processing iterative applications. EX1003, ¶60.

Third, Lippert recognizes that it has long been known to outsource portions of computation tasks "of high resource requirements" to accelerators (boosters), while other portions of the computation tasks are executed by computation nodes. EX1004, [0003], [0028]-[0029] (describing how computation nodes and boosters are "typically" configured with processors suited for different tasks), [0068], [0084]-[0086]. Budenske similarly recognizes the varying requirements of different computations of a given computation task, and that "[h]eterogeneous parallel architectures are ideal computing platforms for efficiently handling computational tasks with such diverse requirements." EX1005, 390 (discussing a computation task that involves "numeric computation, quasi-symbolic computation …, and symbolic computation"). A POSITA would have recognized that Lippert and Budenske follow the same principles of efficient processing of a task, comprising a plurality of sub-tasks, with a heterogeneous computing system, by assigning sub-tasks to processors (e.g., computation nodes or boosters) best suited to execute the sub-task. A POSITA thus would have been motivated to implement Budenske's teachings with a heterogeneous computing system including boosters, as taught by Lippert, partly because "the boosters are implemented to process specific problems at high speed." EX1004, [0028]; EX1003, ¶61.

Fourth, a POSITA reading Lippert would have understood the need to account for sub-task assignment among nodes for iterative applications, which is taught by Budenske. While Lippert focuses on the configuration of the processing resources of the system itself, Lippert only broadly and generally discusses the nature of the "application" or "computation task" being computed. *E.g.*, EX1004, [0015]-[0016]. Further, Lippert discloses dynamically assigning boosters to computation nodes during runtime, and outsourcing parts (i.e., sub-tasks) of the computation tasks to the assigned boosters, suggesting that sub-tasks are also dynamically remapped to newly assigned boosters during processing of the task and "in dependency of processing a specific computation task." *Id.*, [0013] ("[The resource manager may] establish a dynamic assignment at runtime, which means during processing of the computation task"), [0015] ("[B]ooster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"). Budenske, meanwhile, focuses on applications where sub-tasks are iteratively and concurrently executed, and where the processing requirements for a sub-task may evolve over time. EX1005, 387, 390. To that end, Budenske teaches monitoring dynamic parameters of the application during each iteration—where these dynamic parameters relate to the processing of the input data and "have the most impact on the execution time of the subtasks in the application"—and adjusting the assignment of sub-tasks for the next iteration accordingly based on the dynamic parameters. *Id.*, 391, 393. A POSITA thus would have naturally turned to Budenske when implementing Lippert for iterative applications, in order to "assign each … subtask to the processors where it is best suited for execution," and thus minimize execution time as the processing requirements of the input data and/or computation task evolves over time. *Id.*, 390; EX1003, ¶62.

Fifth, a POSITA would have recognized the compatibility of Lippert and Budenske's teachings. Lippert, for example, recognizes that a computation task can be divided into at least a first part for computing by computation nodes, and a second part for computation by boosters. EX1004, [0009]-[0014], [0016]-[0020], [0068]. Further, code blocks (e.g., sub-tasks) are classified as scalable or complex. *Id.*, [0084]. A POSITA would have understood that the "first part of the computation task" computed by the computation nodes includes the less scalable, complex sub-tasks, while the "second part of the computation task" computed by the booster(s) includes the highly scalable sub-tasks. *Id.*; EX1014, 5. Further, Lippert recognizes that "[e]ach code has highly scalable and less scalable complex elements," and so a POSITA would have been motivated to change which sub-tasks are assigned to a first part (computed by computation nodes) and a second part (computed by boosters), regardless of a sub-task's initial classification as complex or scalable (and thus initial mapping to computation nodes or boosters), if information relating to the processing of the sub-task in a preceding iteration suggests that the sub-task would be better processed by a different node. EX1003, ¶63.

Sixth, nothing in these references teaches away from this combination. On the contrary, a POSITA would have recognized that the heterogeneous computing systems of Budenske and Lippert are similar. Specifically, Budenske discloses a heterogeneous computing system including DSP (digital signal processors) and RISC (reduced instruction set computer) processors (EX1005, 391), and a POSITA would have recognized that the DSPs are processors that function as accelerators/boosters and the RISC processors as computing nodes, much like the processing elements in Lippert. A POSITA thus would have understood that the teachings of Lippert and Budenske are compatible and would work well together. EX1003, ¶64.

Finally, a POSITA would have reasonably expected to succeed in implementing the Lippert-Budenske combination because doing so would only entail providing software code, which was well within the skill of a POSITA. *E.g.*, *Keynetik, Inc. v. Samsung Elecs. Co.*, No. 2022-1127, 2023 WL 2003932, at *2 (Fed. Cir. Feb. 15, 2023) (citing *Fonar Corp. v. Gen. Elec. Co.*, 107 F.3d 1543, 1549 (Fed. Cir. 1997)) ("Normally, once the function to be performed by software has been identified, writing code to achieve that function is within the skill of the art."). Further, a POSITA would have reasonably expected success implementing the teachings of Budenske in a heterogeneous computing system as taught by Lippert partly because there is nothing incompatible between the two approaches, and because Budenske explains that its teachings "can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the application and platforms considered here." EX1005, 404. Other prior art references available to the POSITA disclosed successful implementations of Budenske's teachings and confirm that Budenske's approach to re-mapping sub-tasks is feasible and achieves faster execution times compared to other dynamic task scheduling techniques. EX1007, 78-79 (describing Budenske), 92 (illustrating execution times of Budenske's "On-Off" approach compared to other dynamic scheduling approaches for task graphs of various sizes and types); EX1008, 156-65 (discussing Budenske's approach and performance results). In view of these references, a POSITA would have reasonably expected to succeed implementing Lippert-Budenske and achieving the benefits described in each reference. EX1003, ¶65.

Further, the '442 patent provides only a high-level description of adjusting the distribution of sub-tasks between computation nodes and boosters. *E.g.*, EX1001, 3:59-4:3, 4:23-31. This level of disclosure reflects the state of the art, as "it assumes anyone desiring to carry out the process would know of the equipment and techniques to be used, none being specifically described," *In re Fox*, 471 F.2d 1405, 1407 (C.C.P.A. 1973), and thus confirms that POSITAs would have reasonably expected to succeed implementing Lippert-Budenske. EX1003, ¶66.

So combined, the Lippert-Budenske combination dynamically distributes sub-tasks of a computation task among computation nodes and boosters of a heterogeneous computation system between iterations of the computation task, in order to most efficiently process the computation task, and thus renders all challenged claims obvious. EX1003, ¶67.

1. Claim 1

a) Element [1.1] [2]

[1.1] A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:

Lippert-Budenske discloses and teaches this element[3]. Lippert and Budenske each discloses a method of operating a heterogeneous computing system to compute

[2] Reference numbers in the format of [claim#.element#] are added throughout for ease of reference.

[3] Petitioner uses the terms "limitation" and "element" loosely throughout the Petition and, unless otherwise noted, without taking positions on whether the claim language is entitled to patentable weight or is merely part of the claim's environment.

a computation task. Lippert discloses operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, with at least one of the nodes being arranged to compute a computation task comprising a plurality of sub-tasks. Specifically, Lippert discloses a "computer cluster arrangement for processing a computation task," where the "computer cluster arrangement" comprises a plurality of computation nodes and a plurality of boosters (booster nodes). EX1004, [0009]-[0013]. Processors applied in computation nodes are different from the processors applied in boosters. *Id.*, [0029]. A POSITA thus would have understood that the computer cluster arrangement is a heterogeneous computing system because the computer cluster arrangement comprises different types of processors, arranged as computation nodes and booster nodes. Figure 2 of Lippert, reproduced below, "shows a computer cluster arrangement comprising a cluster C as well as a booster group BG," where the cluster as depicted comprises "four computation nodes, also referred to as CN, as well as three boosters, also referred to as B." *Id.*, [0067].

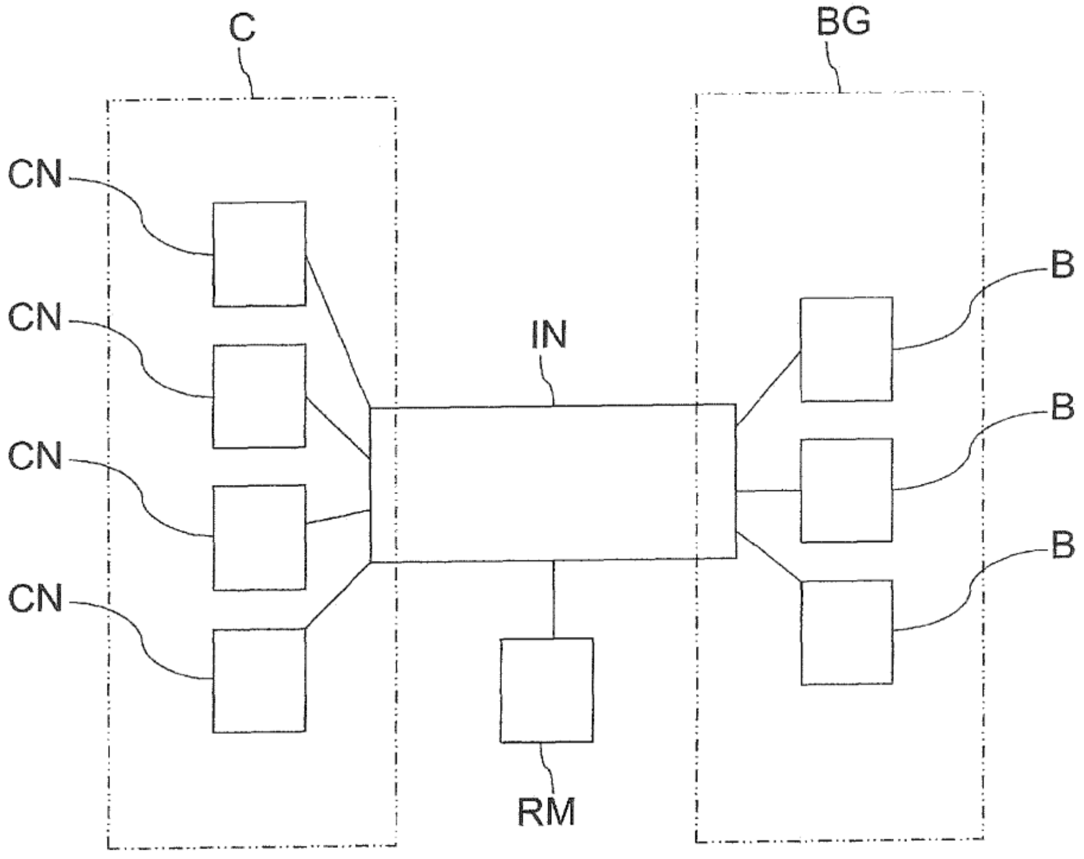


Fig.: 2

EX1004, FIG. 2; EX1003, ¶68.

Lippert discloses that "the computation task may comprise several sub problems, also referred to as sub tasks, which in their entirety describe the overall computation task." EX1004, [0016]. The computation task may be divided "into several parts," and the computer cluster arrangement is arranged "to solve the parts of the computation task in parallel or in succession." *Id.*, [0016]; EX1003, ¶69. Thus, Lippert (and Lippert-Budenske) discloses this claim element.

Budenske discloses a method for operating a heterogeneous computing system to process a computation task, the computation task comprising a plurality of sub-tasks. Budenske discloses a "methodology for real-time on-line input-data dependent remapping of the application subtasks to the processors in the heterogeneous parallel hardware platform." EX1005, 387. Budenske explains that the configuration of the heterogeneous parallel hardware platform (heterogeneous computing system) depends on the expected needs of the application (task) to be computed, and as an example teaches a heterogeneous parallel hardware platform that includes "up to four different types of processors, and up to a total of 64 processors (of all types combined)." *Id.*, 391; EX1003, ¶70.

As explained by Budenske, an "application … can be modeled as an iterative execution of a set of partially ordered subtasks." EX1005, 390. An "application task" is represented as "a *DDG* (data dependency graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies among subtasks." *Id.*, 391. The DDG is "structured as a directed acyclic graph (*DAG*)." *Id.*; EX1003, ¶71.

In combination, Lippert-Budeske discloses a method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks. EX1003, ¶72.

b) **Element [1.2]**

[1.2] in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and

Lippert-Budenske discloses and teaches this element. Lippert discloses computing a "first part of a computation task" and a "second part of the computation task" (*the plurality of sub-tasks*) by, respectively, "at least two of the plurality of computation nodes CN" (*at least a portion of the plurality of computation nodes*) and "at least one booster B" (*at least a portion of the plurality of booster nodes*). EX1004, [0091]. A POSITA would have understood this computing of the parts of the computation task by the at least two of the plurality of computation nodes and the at least one booster as *assigning and processing* the parts of the computation task *in a first distribution*, at least because Lippert teaches "establish[ing] a static assignment [of a booster to a computation node] at start of a processing of a computation task," where "assignment information [is provided] to the computation nodes for outsourcing parts of the computation tasks from at least one computation node to at least one booster." *Id*., [0013]-[0014]; EX1003, ¶73.

Although Lippert discloses the computation node "outsourcing" parts of the computation task to a booster where the computation node processes the first part in a first step and the booster node processes the second part in a second step (e.g., EX1004, [0091]-[0092]), suggesting that the parts are solved in succession, Lippert discloses also that the system may "solve the parts of the computation task *in parallel* or in succession." *Id.*, [0016] (emphasis added). Further, Lippert does not expressly specifically disclose iterative execution of a computation task, and thus does not expressly specifically disclose a "first computing iteration," but Lippert more generally explains that "the person skilled in the art appreciates that any of the steps can be performed iteratively, in a different order and may comprise further sub steps." *Id.*, [0093]; EX1003, ¶74.

Budenske discloses assigning a plurality of sub-tasks to processing resources of a heterogeneous computing system for processing in a first iteration. For example, Budenske discloses, "[f]or the initial iteration through the set of subtasks" (*in a first computing iteration*), "assign[ing] resources (e.g., processors) to the subtasks." EX1005, 391; *see also id.*, 396-97 (describing how to assign sub-tasks to processors). This "initial mapping" (*first distribution*) of the subtasks (*the plurality of sub-tasks*) to the processing resources is used for a first computing iteration of the application (*computation task*). *Id.*, 395; EX1003, ¶75.

A POSITA would have implemented Lippert-Budenske to, in a first computing iteration of an iterative application, assign the plurality of sub-tasks to at least a portion of the computation nodes and at least a portion of the boosters according to an initial mapping, where the initial mapping maps sub-tasks to processors (e.g., computation nodes or boosters), and where the plurality of sub-tasks are thus processed by their assigned computation nodes and boosters. The POSITA would have been motivated to implement Lippert-Budenske in this way at least because doing so would ensure that sub-tasks are assigned to processing resources "best suited" to compute the algorithms of the sub-tasks while accounting for the initial architecture of computation nodes and boosters, and characteristics of the application itself (e.g., what algorithms will be used to compute each sub-task; what are the data dependencies between sub-tasks; etc.). *E.g.*, EX1005, 392, 396-97. Lippert-Budenske thus teaches and renders obvious, in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution. EX1003, ¶76.

c) <u>Element [1.3]</u>

[1.3] generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

Lippert-Budenske discloses and teaches this element. After processing of the sub-tasks with the "initial mapping," Budenske teaches "monitor[ing] the run-time values of the dynamic parameters at the end of each iteration through the underlying

DDG to make a heuristically-based decision whether to continue with the current mapping, or to select and instantiate a new mapping (for the next iteration)." EX1005, 395. Budenske thus discloses using values of dynamic parameters (*information relating to the processing of the sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes*) to generate a "new assignment" (*further distribution*) of sub-tasks to resources "to use for the next execution iteration through the subtasks" (*in a further computing iteration*). *Id.*, 391; EX1003, ¶77.

Specifically, "[a]fter each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image stream being processed." EX1005, 391. The dynamic parameters are "computed by the application as it executes." *Id.*, 393. The values for the dynamic parameters are thus obtained after processing the sub-tasks in an iteration: "After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel." *Id.*, 391. These values are an example of the claim's "information relating to the processing of the plurality of sub-tasks" for at least two reasons. First, they affect the nature of the sub-tasks being processed. For example, as dynamic parameters change, the computations of the sub-tasks change. *E.g.*, *id.*, 393 (explaining that dynamic parameters "change during run time" and "have the most impact on the execution time of the subtasks in the application"). The dynamic parameters therefore relate to the processing of the sub-tasks. Second, they are determined by the processing of the sub-tasks. *Id.* (explaining that dynamic parameters are "computed by the application as it executes"). Therefore, the result of processing the sub-tasks—the values of the dynamic parameters—is information relating to the processing of the sub-tasks. The "most recent values of such dynamic parameters" are then used to determine whether "to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration," and "[i]f it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks." *Id.*, 391. In particular, Budenske teaches generating a "vector of *D* representative values" as an "approximation of the set of current actual dynamic parameters," and using that vector to identify the optimal mapping of sub-tasks from the possible mappings (tracked in a "Mapped DDG" or "MDDG Table"). *Id.*, 402. Budenske thus teaches selecting a new mapping from a set of candidate mappings that best matches the values of the dynamic parameters (*the information*), and instantiating (*generating*) the further distribution according to the selected mapping for the next computing iteration. *Id.*, 402-03. Budenske also discloses an alternative wherein a fully "on-line heuristic" is used, such that the new

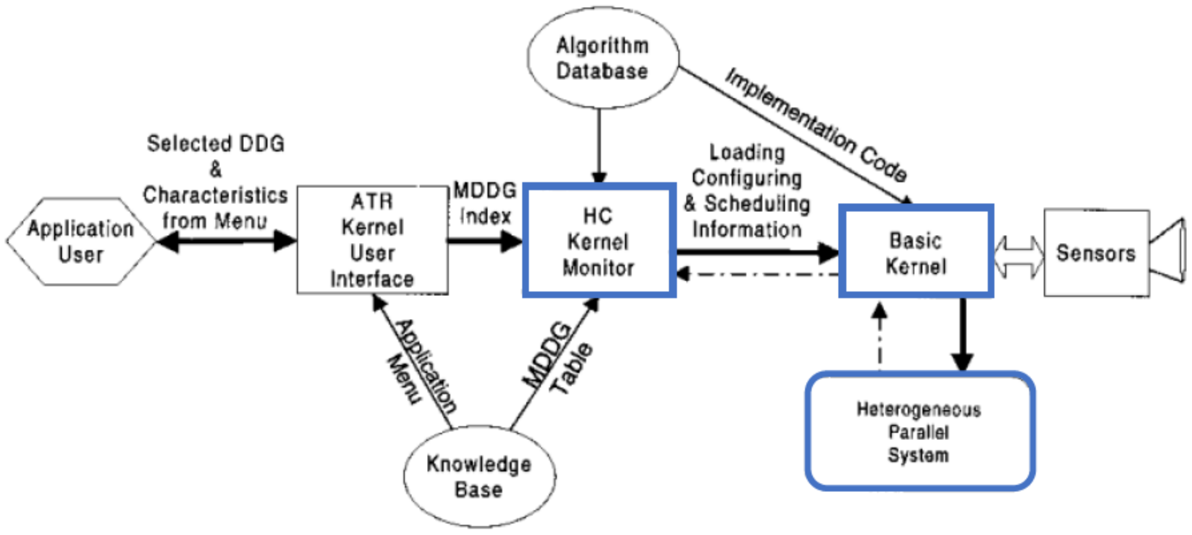
mapping is generated in real-time between iterations based on the information. *Id.*, 389-90, 398, 403; EX1003, ¶78.

A POSITA would have found it obvious to implement the heterogeneous computing system of Lippert with the dynamic sub-task mapping method of Budenske such that, between iterations, Lippert-Budenske uses values of dynamic parameters calculated during a preceding iteration to generate a new assignment of sub-tasks to the plurality of computation nodes and the plurality of boosters for processing in the next iteration. The POSITA would have been motivated to implement Lippert-Budenske in this way partly in order "to reduce execution time of the next iteration." EX1005, 391. The POSITA would have further been motivated to implement Lippert-Budenske in this way because Budenske explains that, "[b]ecause the execution time of application subtasks … is highly input-data dependent …, this matching and scheduling of application subtasks to processors *must* be performed dynamically at run time." *Id.*, 390 (emphasis added). The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske partly because Lippert and Budenske each provide ample implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. EX1003, ¶79. Lippert-Budenske thus teaches and renders this element and this claimed method obvious.

2. Claim 2

2. The method according to claim 1, wherein an application manager receives the information and determines the further distribution.

Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske renders claim 1 obvious, as discussed *supra* Section VII.C.1. Budenske discloses an "HC Kernel" (also referred to as "HC Kernel Monitor") and a "Basic Kernel" that cooperate to manage an application, and thus a POSITA would have understood aspects of the HC Kernel and the Basic Kernel together as an *application manager*. EX1005, 388-89, 401-02. According to Budenske, "the HC Kernel interacts with the Basic Kernel … to execute the application and monitor its execution so it can decide when to remap the subtasks onto the hardware platform" (*id.*, 388), where the Basic Kernel receives "the latest values of … dynamic parameters" at the end of each iteration (*receives the information*). *Id.*, 402 (describing how the Basic Kernel "begin[s] execution of the application task"). The Basic Kernel sends the latest values of the dynamic parameters to an HC Kernel Monitor, which then "use[s] the most recent values of these dynamic parameters to estimate if changing the matching and scheduling will reduce the expected execution time of the next iteration through the corresponding DDG." *Id.* Figure 3 of Budenske depicts the Basic Kernel and the HC Kernel Monitor (together, *the application manager*) interacting with the heterogeneous parallel computing system during execution of an iterative application:



Id., FIG. 3 (annotated); *see also id.*, FIG. 1; EX1003, ¶80.

A POSITA would have understood the HC Kernel and the Basic Kernel of Budenske together as an "application manager" that "receives the information and determines the further distribution" because the Basic Kernel receives the values of the dynamic parameters (*receives the information*) from the heterogeneous parallel computing system and inputs the dynamic parameter values to the HC Kernel to determine a new assignment of sub-tasks to processing resources (*determines the further distribution*). EX1005, 391, 402. Lippert-Budenske thus discloses and teaches this claim element and renders this claim obvious. EX1003, ¶81.

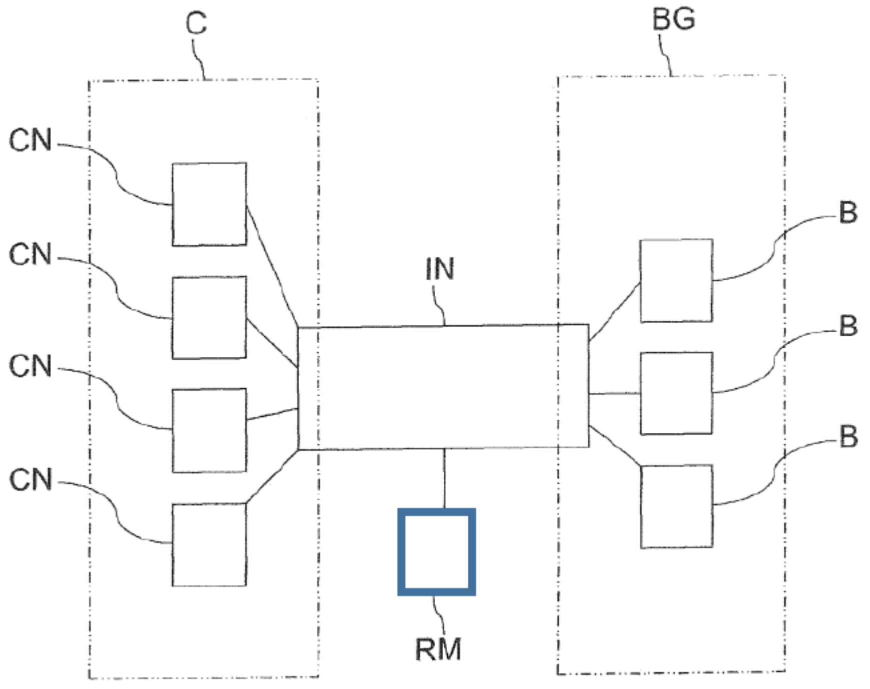
3. Claim 3

a) Element [3.1]

[3.1] The method according to claim 2, wherein a resource manager determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task and

Lippert-Budenske discloses and teaches this claim element and renders this claim obvious for the foregoing and following reasons. Specifically, as discussed below, Lippert discloses a "resource manager" and Budenske discloses an HC Kernel Monitor, and a POSITA implementing Lippert-Budenske would combine aspects of Lippert's resource manager and Budenske's HC Kernel Monitor into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task. Lippert-Budenske thus includes a *resource manager* that manages resources of the heterogeneous computing system (e.g., computation nodes, booster nodes, and the assignment thereof for processing a computation task), and an *application manager* that manages applications (e.g., computation tasks) processed by the heterogeneous computation system, where the resource manager handles an initial mapping of sub-tasks because it manages the initial arrangement of the resources and does not depend on information from processing the sub-tasks, whereas the application manager handles re-mapping of the sub-tasks to the resources because it receives information relating to the processing of the sub-tasks during an application's execution. *Supra* Section VII.C.2; EX1003, ¶82.

Lippert discloses a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes as a function of the computation task. The resource manager "establish[es] a static assignment at start of a processing of a computation task," and "establish[es] a dynamic assignment at runtime, which means during processing of the computation task." EX1004, [0013], [0014], [0017]. FIG. 2 of Lippert depicts the resource manager RM of the heterogeneous computing system.



EX1004, FIG. 2 (annotated). This resource manager RM is similar to the resource manager RM of the '442 patent. *See* EX1001, FIG. 1 (element 28). The *resource manager* of Lippert determines the computation node-booster assignment based on "computation task requirements," and thus *as a function of the computation task*. EX1004, [0017]. Specifically, "booster allocation is performed *as a function of application needs*, which means in dependency of processing a specific computation task." *Id.*, [0015] (emphasis added). The resource manager "determin[es] an initial assignment of boosters to computation nodes" based on a predetermined assignment metric, and dynamically assigns boosters to computation nodes "at runtime" as the computation task requirements evolve and the assignment metric is accordingly altered. *Id.*, [0017], [0013], [0030]. The assignment metric is managed by the resource manager and is updated "as a function of a load balancing, which detects workload of the computer cluster arrangement, especially of the boosters." *Id.*, [0017]. For the assignment metric, Lippert teaches "detect[ing] computing capacities of boosters and furthermore detect[ing] computation task requirements," so that a booster assigned to a computation node "provides the required capacities to the computation node." *Id.* A POSITA would have understood the resource manager's assignment based on "computation task requirements" as being determined *as a function of the computation task*. *Id.*, [0017]; EX1003, ¶83.

Budenske, meanwhile, teaches an HC Kernel Monitor that determines the assignment of the plurality of sub-tasks to the plurality of nodes for the first computing iteration as a function of the computation task. Specifically, Budenske teaches an HC Kernel Monitor that "establish[es] the initial mapping" of sub-tasks to processing resources for an initial iteration (*the assignment of the plurality of sub-tasks to the plurality of computation nodes … for the first computing iteration*). EX1005, 402. The initial mapping is an MDDG for a given initial scenario (e.g., set of dynamic parameter values), where the MDDG is structured as a directed acyclic graph that arranges sub-tasks according to "data dependencies among subtasks" with algorithms selected for implementing the sub-tasks (*as a function of the computation task*). *Id.*, 391, 396, 402. A POSITA would have understood the HC Kernel Monitor partly as a *resource manager* because it "decide[s] how to assign resources (e.g., processors) to the subtasks." *Id.*, 391; EX1003, ¶84.

In combination, Lippert-Budenske includes a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes, as taught by Lippert, for the first computing iteration, as taught by Budenske. The assignment is determined as a function of the computation task, as taught by both Lippert and Budenske, because doing so would "enforce data-dependency constraints among the subtasks" (EX1005, 397) and ensure that the sub-tasks of an iterative application are assigned "to the processors where [they are] best suited for execution" (*id.*, 390). *See also* EX1004, [0015], [0017]. EX1003, ¶85.

b) **Element [3.2]**

[3.2] wherein the application manager receives the information and processes the information as input to the resource manager such that the resource manager dynamically alters further distribution during the computing of the computation task.

Lippert-Budenske discloses and teaches this element, and thus renders the claimed method obvious for the foregoing and following reasons. As discussed *supra* Section VII.C.2, Lippert-Budenske includes an application manager that receives the information (*the application manager receives the information*) and determines the further distribution (*processes the information*). Lippert-Budenske further includes a resource manager that "establish[es] a dynamic assignment at runtime, which means during processing of the computation task" (*dynamically alters further distribution during the computing of the computation task*), as taught by Lippert. EX1004, [0013]; *supra* Section VII.C.3.a). EX1003, ¶86.

A POSITA implementing Lippert-Budenske would have found it obvious to input the new mapping of sub-tasks by the application manager into the resource manager such that the resource manager dynamically alters distribution of computation nodes and booster nodes during the computing of the computation task, because doing so would utilize the "maximum flexibility in providing hardware resources" taught by Lippert. EX1004, [0015]; *see also* EX1005, 390 (teaching to "fully utilize the architectural flexibility"). That is, in addition to re-mapping the sub-tasks to processing elements of the system as taught by Budenske, Lippert-Budenske also dynamically assigns computation nodes and boosters to each other "during processing of the computation task" as taught by Lippert (EX1004, [0013]). By providing the new mapping for the next iteration to the resource manager, Lippert-Budenske would, for example, allocate additional boosters for processing a given sub-task or otherwise re-configure computation node-booster assignment in order to implement the new mapping and "reduce execution time of the next iteration" (EX1005, 391). EX1004, [0014]-[0017], [0068]. In this way, Lippert-Budenske "provide[s] fault tolerance in case of accelerator failures" and avoids "a lack of resources or … an excessive supply of resources" when executing an application. EX1004, [0003]. EX1003, ¶87.

A POSITA would have understood that "the resource manager dynamically alters further distribution" as recited in this claim element *encompasses* "the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other," because claim 4 depending from claim 3 specifies this, and further because the '442 patent describes the resource manager adjusting booster assignments according to updated assignment information from the application manager. EX1001, 4:27-31. Thus, in view of the above, Lippert-Budenske discloses and teaches this element and this claim. EX1003, ¶88.

4. Claim 4

4. The method according to claim 3, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. As discussed *supra* Section VII.C.3, the resource manager of Lippert-Budenske receives the information and, if appropriate, dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. EX1004, [0013], [0017]; EX1003, ¶89.

5. Claim 5

5. The method according to claim 1, wherein daemons operate in the plurality of computation nodes and the plurality of booster nodes to generate the information.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons addressing claim 1 and for the following reasons. Lippert-Budenske renders claim 1 obvious. *Supra* Section VII.C.1. Lippert discloses "detect[ing] computing capacities of boosters" (EX1004, [0017]), and Budenske teaches values of dynamic parameters being reported to the Basic Kernel during each iteration (*e.g.*, EX1005, 402). In order to send the dynamic parameter values taught by Budenske, as well as information regarding computing capacities as taught by Lippert, from each computation node and each booster node to a central manager to determine a re-mapping of sub-tasks, a POSITA thus would have implemented Lippert-Budenske with daemons at each computation node and each booster node configured to collect and send this information. POSITAs knew that a daemon is a background process that performs a specific task, and that daemons were commonly implemented at compute nodes to execute tasks at the compute nodes and report status information back to a central resource manager for managing cluster resources. *E.g.*, EX1013, 5-7 (describing a central controller daemon and daemons running at each compute node "to export control to SLURM"), 20 (describing a daemon at each node "for managing user jobs and monitoring system state," where its "most common action … is to report system state on request"); EX1021, 7:66-67 (defining a "daemon" as "a program that runs unattended to perform a standard service"), 8:2-3 ("Daemon processes generally provide services that must be available at all times."), 2:54-3:18 (describing using daemons to facilitate communication in a "parallel processing environment" comprising "a plurality of nodes connected through a network to one another"). A POSITA thus would have been motivated to implement background processes (i.e., daemons) operating at each node that communicate the relevant information in a status update for dynamically remapping the sub-tasks among the nodes, to beneficially facilitate inter-element communication while providing communication authentication, and to enable independent control of task execution and reporting of task processing at each node. *E.g.*, EX1012, 9 (describing a central daemon and node daemons or "TaskTrackers" for executing tasks at the node), 38-40 (describing a "NodeManager" daemon at each node "responsible for launching the applications' containers, monitoring their resource usage …, and reporting the same to the ResourceManager"); EX1013, 10 (describing how daemons authenticate and secure communications between nodes); EX1022, 3:63-66 ("As is commonly known, a daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events."), 3:66-4:14 (explaining that "[t]ypical daemon processes … perform administrative tasks" and are implemented in "any environment … which is capable of implementing background processes and supports communications between the various Nodes of the Cluster(s)"); EX1003, ¶90.

6. <u>Claim 6</u>

6. The method according to claim 1, wherein the first distribution is determined based on a rating provided in source code for each sub-task in the plurality of sub-tasks.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the above reasons associated with claim 1 and the following reasons. Lippert discloses determining the first distribution based on a rating provided in source code for each sub-task in the plurality of sub-tasks. EX1003, ¶91.

Regarding *a rating*, Lippert discloses priority, complexity, and scalability information (individually and together, "*a rating*") for each sub-task because the sub-tasks comprise the priority information, complexity information, and scalability information. Specifically, Lippert discloses that "[s]pecific *computation tasks may comprise priority information*, which indicates how urgently this specific computation task has to be computed." EX1004, [0020] (emphasis added). This priority indicates "how urgent a processing of a computation task, or at least a part of a computation task, is compared to other parts of computation tasks being originated from other computation nodes." *Id.* This relates to the dependency between sub-tasks, where the processing of one sub-task depends on the processing of another sub-task. Regarding complexity information and scalability information, Lippert discloses applications or computation tasks where "[e]ach code has highly scalable and less scalable complex elements," broken down into "Exascale Code Blocks" and "complex Code Blocks," respectively (i.e., scalable sub-tasks and less-scalable complex sub-tasks). *Id.*, [0084]. As "the distinction between highly scalable and complex is made on the level of code blocks," the code blocks (sub-tasks) include the complexity information and scalability information. *Id.* Each sub-task of the computation task thus comprises priority information, complexity information, and scalability information (*a rating*). EX1003, ¶92.

The rating is *provided in source code for each sub-task in the plurality of sub-tasks* because the computation task "comprises" the information and is "defined by means of … a source code." EX1004, [0016]; EX1003, ¶93.

Regarding *the first distribution is determined based on a rating*, Lippert further discloses a "predetermined assignment metric" for assigning nodes to each other, with parts of the computation task allocated accordingly, being specified as a function of "complexity information, scalability information," and "priority information" (*a rating*). EX1004, [0023], [0018], [0043]. This information is specific to each sub-task, and the assignment in Lippert, including the initial distribution of sub-tasks among the nodes, is based partly on this information. *E.g.*, *id.*, [0015] ("[B]ooster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task."), [0017]-[0020] (describing using the assignment metric to assign nodes). Lippert teaches adapting the architecture of the computation nodes and the boosters of a computing cluster to account for the scalability of the code blocks. *Id.*, [0084]-[0088]; EX1003, ¶94.

A POSITA would have understood that Lippert thus discloses and teaches determining the first distribution of sub-tasks based on a rating provided in source code for each sub-task, including a rating relating to priority, complexity, and scalability, to ensure that the part of a computation task distributed to a booster is suitable for the resource capacities of the booster. This is consistent with the disclosure in the '442 patent of an operator estimating a scalability factor for each sub-task and distributing the sub-tasks accordingly. EX1001, 4:16-47. Further, the '442 patent confirms that POSITAs already knew how to annotate application code to indicate that a given task is "highly scalable" and thus suitable for execution by boosters. EX1001, 1:39-55 (citing EX1014); EX1014, 15-16 (prior art describing how to designate portions of code as sub-tasks as highly scalable and as suitable for execution on specified hardware devices, e.g., boosters); EX1003, ¶95.

A POSITA would have found it obvious when implementing Lippert-Budenske to determine the initial mapping of sub-tasks to computation nodes and boosters based on a rating provided in source code for each sub-task, where such rating comprises an indication of priority, complexity, and scalability, as taught by Lippert. The POSITA would have been motivated to do so to create optimal assignments of sub-tasks to processing resources (e.g., computation nodes or boosters) that account for suitability of the processing resource for the corresponding scalability/complexity/priority of the sub-task while minimizing the expected execution time. The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske in this way because Lippert provides implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. EX1003, ¶96. Lippert-Budenske thus renders this claim obvious.

7. <u>Claim 7</u>

7. The method according to claim 1, wherein the information is used to provide a grouping of sub-tasks in at least one of the first computing iteration and the further computing iteration.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons associated with claim 1 and the following reasons. Regarding *a grouping of sub-tasks*, Lippert discloses "divid[ing] the computation task into several parts," including a first part computed by computation node(s) and a second part computed by booster(s). EX1004, [0016], [0009]-[0012]. A POSITA would have understood these "parts" of the computation task as groupings of sub-tasks, e.g., a first grouping of sub-tasks for processing by computation nodes and a second grouping of sub-tasks for processing by boosters, because Lippert discloses that a computation task "comprise[s] several sub problems, also referred to as sub tasks." *Id.*, [0016]; EX1003, ¶97.

As discussed *supra* Section VII.C.1.c), Lippert-Budenske discloses and teaches using information relating to the processing of the plurality of sub-tasks to generate a further distribution of the sub-tasks for a further computing iteration. In view of this disclosure and teaching, a POSITA implementing Lippert-Budenske would have been motivated, with a reasonable expectation of success, to use the values of the dynamic parameters (*the information*) from a preceding iteration to provide, via the new mapping of sub-tasks, a first grouping of sub-tasks and a second grouping of sub-tasks for processing respectively with computation node(s) and booster(s) in the next iteration (*the further computing iteration*), at least because Lippert expressly discloses providing such groupings of sub-tasks. EX1004, [0016], [0009]-[0012]; EX1003, ¶98.

8. <u>Claim 8</u>

8. The method according to claim 3, wherein a daemon operating at a node generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the reasons associated with claims 1-3 and 5, and the following reasons. Lippert-Budenske renders claim 3 obvious, and further renders the daemons operating at the plurality of computation nodes and booster nodes obvious. *Supra* Sections VII.C.3, VII.C.5. Lippert discloses "detect[ing] workload of the computer cluster arrangement, especially of the boosters," as well as "computing capacities." EX1004, [0017]. For Lippert's resource manager to detect such information, a POSITA would have understood that a background process (i.e., a daemon) implemented at each node to measure and report workload and computing capacity information (*generates a measure of a loading of the node during processing of a* *sub-task of the plurality of sub-tasks*) would be the most common way to generate and communicate such information, and the easiest to implement because daemons already exist for performing this function. *E.g.*, EX1012, 38 (describing a per-node daemon "monitoring their resource usage (CPU, memory, disk, network) and reporting the same to the ResourceManager"); EX1013, 6-7 (describing a "daemon running on each compute node"), 19-20 (describing how node daemons provide "resource consumption information … associated with a job"); EX1003, ¶99.

A POSITA thus would have been motivated, with a reasonable expectation of success, to implement Lippert-Budenske with a daemon operating at each computation node and booster node that generates a measure of a loading (workload) of the node during processing of a sub-task of the plurality of sub-tasks, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system, as taught by Budenske, to also account for "load balancing" as taught by Lippert. EX1004, [0017]; EX1003, ¶100.

9. <u>Claim 9</u>

a) <u>Element [9.1]</u>

[9.1] A heterogeneous computing system comprising:

Lippert-Budenske discloses and teaches this element, at least because Lippert and Budenske each disclose a heterogeneous computing system. *Supra* Section VII.C.1.a). EX1003, ¶101.

b) **<u>Element [9.2]</u>**

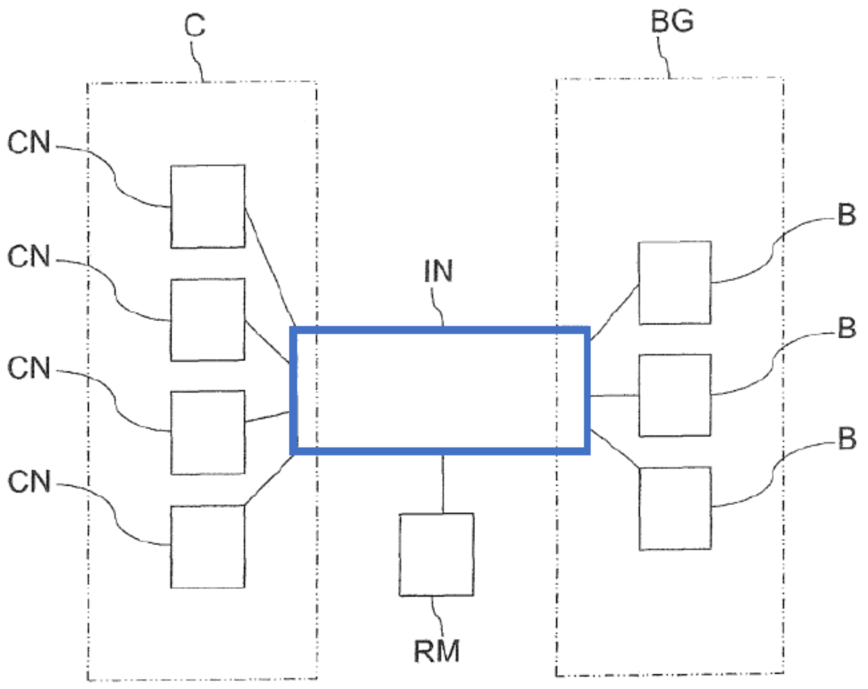
[9.2] a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks;

Lippert-Budenske discloses and teaches this element, at least because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. *Supra* Section VII.C.1.a). EX1003, ¶102.

c) **<u>Element [9.3]</u>**

[9.3] a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes;

Lippert-Budenske discloses and teaches this element, at least because Lippert discloses the plurality of computation nodes and the plurality of booster nodes "interfacing a communication infrastructure." EX1004, [0010]-[0011], [0016]. Thus, "the computation nodes as well as the boosters interact by means of the communication infrastructure." *Id.*, [0016]. FIG. 2 of Lippert depicts the communication infrastructure IN, also referred to as an interconnect, interfacing with each computation node CN and each booster B. *Id.*, [0067], FIG. 2.



EX1004, FIG. 2 (annotated); EX1003, ¶103.

Budenske similarly discloses that "[a]ll the processors of all types will have communication paths to one another," via a "bus [that] can be used to provide communications among different collections of processors." EX1005, 392. EX1003, ¶104.

Thus, when implementing Lippert-Budenske, a POSITA would have found it obvious for the heterogeneous computing system to include a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes, at least because Lippert and Budenske each disclose such a communication interface. EX1003, ¶105.

d) Element [9.4]

[9.4] a resource manager for assigning at least a portion of the plurality of booster nodes and at least a portion of the plurality of computation nodes to each other for the computing of the one or more tasks in a first computing iteration; and

Lippert-Budenske discloses and teaches this element. *Supra* Sections VII.C.3, VII.C.4. EX1003, ¶106.

e) Element [9.5]

[9.5] an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2. Lippert-Budenske thus discloses and teaches all elements and renders this claimed system obvious. EX1003, ¶107.

10. Claim 10

10. The heterogeneous computing system according to claim 9, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other.

Lippert-Budenske discloses and teaches this element, and thus renders this claim obvious. *Supra* Sections VII.C.9, VII.C.4; EX1003, ¶108.

VIII. GROUND 2: CLAIMS 2-5 AND 8-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE AND KAMBATLA

As explained below, claims 2-5 and 8-10 are obvious over Lippert in view of Budenske and Kambatla. While Lippert and Budenske in combination render claims 1-10 obvious, the express teachings of Kambatla regarding resource management daemons reflects the knowledge of a POSITA and further confirms the obviousness of the claims. EX1003, ¶109.

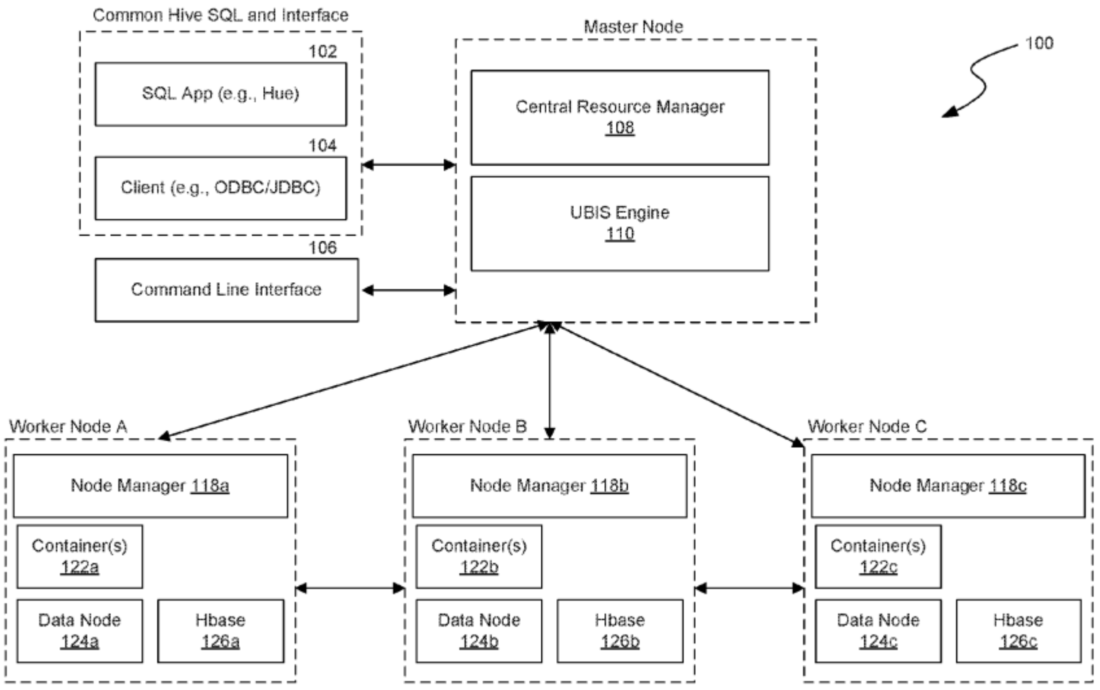
A. Kambatla (EX1006)

Kambatla is a U.S. patent application filed May 15, 2017, and claiming priority to a provisional application filed September 14, 2016, and thus qualifies as prior art under at least Section 102(a)(2). EX1006, Cover. Kambatla is analogous art to the ’442 patent because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1006, [0002], [0004]; EX1003, ¶110.

Kambatla discloses a resource management system for the “allocation and management of computing resources in distributed computing clusters.” EX1006, [0002]. Kambatla teaches techniques referred to as “utilization-based incremental scheduling” (UBIS) that “opportunistically allocate[s] computing resources not utilized by prior allocations.” *Id.*, [0017]. Kambatla proposes UBIS as an improvement to known “cluster scheduler” frameworks such as Apache Hadoop

YARN ("Yet Another Resource Negotiator"), used for scheduling (i.e., assigning) multiple tasks of a job to cluster computing resources. *Id.*, [0004], [0020] ("UBIS can be implemented in or for use with Apache YARN"), [0023]-[0024], [0085]. EX1003, ¶111.

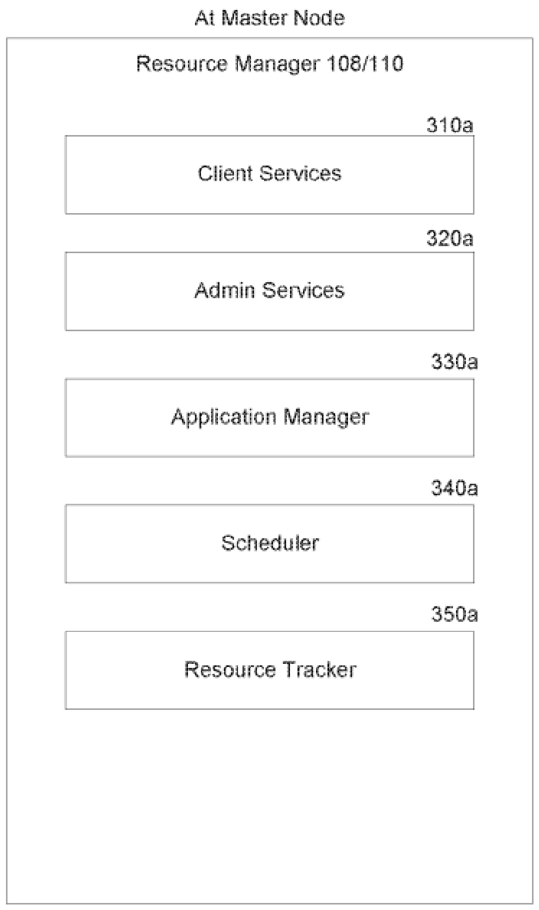
Kambatla discloses a computing environment, depicted in Figure 1, comprising a master node and a plurality of worker nodes in communication, where the master node assigns tasks of a job to the computing resources of the worker nodes for execution. EX1006, [0021]-[0022].



Id., FIG. 1. To facilitate allocation of computing resources for tasks, Kambatla discloses daemons at each computing node, including a central resource manager daemon at the master node and a node manager daemon at each worker node. *Id.*,

[0030]-[0036] (describing resource manager), [0037]-[0041] (describing node managers), FIGS. 3A-B. The resource manager is "configured to manage and arbitrate resources among applications in the system." *Id.*, [0023]. EX1003, ¶112.

Although Figure 1 depicts the resource manager 108 and the UBIS engine 110 as separate components of the master node, Figure 3A of Kambatla discloses a single resource manager comprising both components. EX1006, [0023], [0030], FIG. 3A. The resource manager comprises, *inter alia*, "an application manager 330a, a scheduler 340a, and a resource tracker 350a." *Id.*, [0031].



EX1006, FIG. 3A; EX1003, ¶113.

B. <u>Combining Lippert, Budenske, And Kambatla</u>

A POSITA would have found it obvious to implement a combination of Lippert and Budenske, as discussed *supra* Section VII.C. A POSITA further would have been motivated, with a reasonable expectation of success, to combine the teachings of Lippert-Budenske with Kambatla's resource scheduling daemons. A skilled artisan would have combined these respective teachings into an improved design (hereinafter "Lippert-Budenske-Kambatla") that included benefits from each reference. The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
  - A plurality of computation nodes;
  - A plurality of booster nodes; and
  - A communication infrastructure interfacing each of the computation nodes and the booster nodes;
- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of

the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02); and

- Dynamically assign, during processing of the computation task, computation nodes and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);

- An application manager module, as taught by Kambatla, arranged to:
  - Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);
  - Monitor execution of sub-tasks to collect information relating to processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
  - Assign, according to a new mapping obtained using information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02);
- A node manager daemon at each computation node and booster node, as taught by Kambatla, arranged to:

- Generate information relating to the processing of a sub-task at the node; and
- Report the information to the application manager module.

EX1003, ¶114.

A POSITA would have been motivated, with a reasonable expectation of success, to so combine Lippert, Budenske, and Kambatla by implementing the computer cluster of Lippert with the dynamic re-mapping of sub-tasks between iterations of an iterative application as taught by Budenske, and with the resource management daemons as taught by Kambatla, for at least the following reasons. EX1003, ¶115.

First, Kambatla, like Lippert-Budenske, discloses a heterogeneous computing system that processes a task comprising a plurality of sub-tasks. EX1006, [0004], [0021], [0059]. Kambatla further teaches that such systems enable processing of not just one task or application, but multiple tasks concurrently, with the processing resources of the system being shared by the multiple tasks, and provides techniques for optimally scheduling tasks across the processing resources. *Id.*, [0004], [0017]. A POSITA thus would have been motivated to incorporate the complementary teachings of Kambatla into Lippert-Budenske in order to better leverage the full processing capacity of the heterogeneous computing system. EX1003, ¶116.

Second, a POSITA implementing Lippert-Budenske would have recognized the need to facilitate inter-element communication in a way that allows the resource manager to efficiently allocate sub-tasks to nodes, and Kambatla teaches how to do so. For example, while Lippert discloses the computation nodes, booster nodes, and resource manager communicating with each other via the communication infrastructure (*e.g.*, EX1004, [0016]), Kambatla describes a node manager daemon operating at each node to communicate with a central resource manager daemon and to manage the execution of tasks at the node. EX1006, [0023], [0038]-[0041]. A POSITA thus would have naturally turned to the teachings of Kambatla for implementation details on communicating information (i.e., values of dynamic parameters, processing capacities of the node) to the central resource manager. EX1003, ¶117.

Third, the particular teachings of Kambatla follow the same principles as Lippert-Budenske regarding the utilization of processing resources. For example, Kambatla describes the prior art background of allocating cluster computing resources to "jobs [that] typically include multiple tasks," and the problem of "resource fragmentation and severe under-utilization of the computing resources in the cluster." EX1006, [0003]-[0005]; *see also* EX1004, [0005] (discussing the problem of "over- or under subscription of accelerators Acc depending on the computation task"). This occurs because "[t]he amount of computing resources required to process a given task can be difficult to predict," as "[i]t is inevitably difficult to accurately estimate the resource requirements of a job or its constituent tasks because: (i) resource usage of a task varies over time, and (ii) resource usage can vary across tasks of the same job based on the input they process." EX1006, [0005]. Kambatla addresses this problem by monitoring "actual utilization of resources at a computing node" via a YARN-based resource manager that communicates with node managers at nodes of a cluster, and "improv[ing] effective resource utilization in a distributed computing cluster." EX1006, [0017], [0042]; *id.*, [0018]-[0023], [0030]-[0036] (describing a central resource manager daemon), [0037]-[0041] (describing a node daemon), [0042]-[0049] (describing "Optimizing Resource Utilization"). This utilization-based resource allocation for applications deployed to a computer cluster is complementary to the dynamic assignments of Lippert-Budenske. *E.g.*, EX1004, [0015] (teaching "booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"); EX1005, 390-91 (describing "input-data dependent" execution of sub-tasks and the remapping of sub-tasks to processors as dynamic parameters evolve over time). Thus, a POSITA seeking to fully leverage the processing capacity of a computer cluster, for example by processing multiple iterative applications concurrently, would naturally seek out and incorporate teachings on cluster resource management daemons that facilitate dynamic resource allocation, such as those taught by Kambatla. EX1003, ¶118.

Finally, a POSITA had a reasonable expectation of success implementing Lippert-Budenske-Kambatla, at least because a POSITA was familiar with and understood how to implement "known resource management techniques (such as in YARN)" (EX1006, [0023]), and because doing so would involve writing only minor software code that was well within the skill of a POSITA. *E.g.*, *Keynetik*, 2023 WL 2003932, at *2 (citing *Fonar Corp. v. Gen. Elec. Co.*, 107 F.3d 1543, 1549 (Fed. Cir. 1997)); *see also* EX1012, xviii (explaining that "YARN makes the effort of building [distributed computing] systems significantly simpler"). Further, a POSITA would have reasonably expected to succeed incorporating Kambatla's teachings into Lippert-Budenske because Kambatla explains that its teachings can be successfully implemented in heterogeneous computing systems comprising "CPU and memory resources" (i.e., computation nodes) and "GPU[s] (graphical processing [units])" (i.e., boosters). EX1006, [0020], [0023], [0059]. Further, the '442 patent admits that the use of daemons at nodes to facilitate communication was already known, confirming that a POSITA knew how to implement such features. EX1001, 3:6-18, 3:39-43, 4:32-37; *In re Fox*, 471 F.2d at 1407; EX1003, ¶119.

Thus, a POSITA implementing the dynamic assignment of sub-tasks between iterations, as taught by Budenske, to computation nodes and boosters of a heterogeneous computing system, as taught by Lippert, would have been motivated to implement a cluster scheduling framework as taught by Kambatla, in order "to improve effective resource utilization" and thus minimize "processing duration." EX1006, [0042]; EX1003, ¶120.

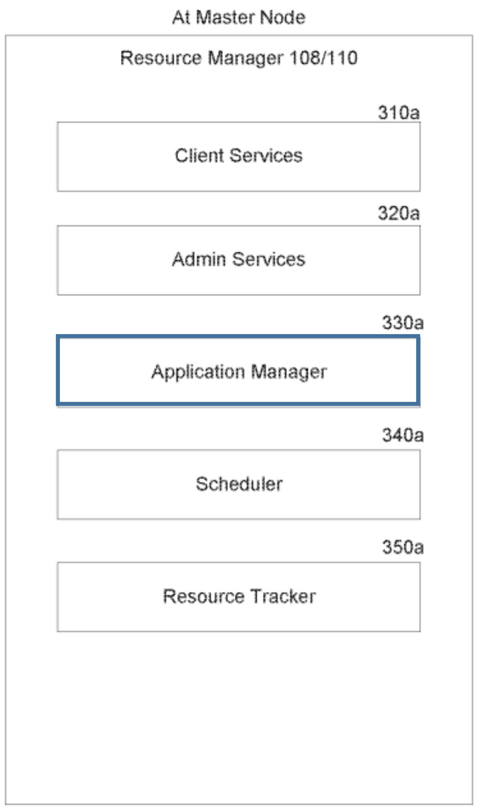
1. **Claim 2**[4]

Lippert-Budenske-Kambatla discloses and teaches this element and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske renders claim 1 obvious, as discussed *supra* Section VII.C.1. Further, Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2; EX1003, ¶121.

Moreover, Kambatla expressly discloses various modules of a central resource manager that a POSITA would have understood as the "application manager" recited in the claim. Kambatla discloses an "application manager 330a … responsible for maintaining a collection of submitted 'applications' or 'jobs.'" EX1006, [0034]. The application manager is a module of a "resource manager daemon." *Id.*, [0031]. The application manager includes an "application master service responsible for communicating with all the application masters" at the

[4] The full claim language is included in mapping the claims for Ground 1 (*supra* Section VII) and in the Listing of Claims (EX1020). Petitioner hereinafter does not reproduce the claim language, but instead references it by element/claim number.

computing nodes. *Id.*, [0034]. FIG. 3A of Kambatla depicts the resource manager 108 including the application manager 330a.



Id., FIG. 3A (annotated). Further, the resource tracker 350a of the resource manager "monitors available resources at the nodes, by receiving status updates from the worker nodes," while "[t]he scheduler 340a is responsible for allocating resources to the various applications." EX1006, [0035]-[0036]; *id.*, [0058] (describing how the resource manager "can actively monitor resource usage of each container at the worker nodes and of the worker nodes themselves," via "heartbeat information" that includes "utilization information"). Despite being depicted as separate components, Kambatla teaches that the resource manager can "include more or fewer components, organized in various configurations." EX1006, [0030]. A POSITA thus would have understood that the functions of the application manager 330a, the scheduler 340a, and the resource tracker 350a were combinable into a single component (*application manager*) that manages an application deployed to cluster resources, communicates with the resources to monitor the status of the application, and assigns sub-tasks of the application to the resources for processing. EX1003, ¶122.

In combination, a POSITA implementing Lippert-Budenske-Kambatla would have found it obvious to use an application manager as taught by Kambatla to perform the functions of the HC Kernel and Basic Kernel of Budenske as discussed *supra* Section VII.C.2, at least because the application manager and the Kernels similarly manage the execution of an application on a plurality of computing devices. Further, a POSITA would have been motivated to implement the dynamic assignment of sub-tasks to the compute nodes and booster nodes of a heterogeneous computing system, as taught by Lippert-Budenske, with the application manager of Kambatla in order to efficiently enable and manage the execution of multiple such applications on the plurality of computing resources. *E.g.*, EX1006, [0004], [0023]-[0024], [0034]; EX1003, ¶123.

The POSITA would have reasonably expected to succeed because implementing such an application manager would entail only providing simple software code and configurations to perform the known functions, which was within the skill of a POSITA. *E.g.*, EX1006, [0101] (explaining broadly how "the routines executed to implement the embodiments of the disclosure[] may be implemented"); EX1012, 59-84 (detailing how to install and configure YARN on a cluster); EX1003, ¶124.

2. **<u>Claim 3</u>**

a) **<u>Element 3.1</u>**

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. *Supra* Section VIII.B.1 (claim 2). Lippert-Budenske discloses and teaches this element, as discussed *supra* Section VII.C.3. Further, Kambatla discloses a "general resource manager configured to manage and arbitrate resources among applications in the system." EX1006, [0023], FIGS. 1, 3A. A POSITA implementing Lippert-Budenske-Kambatla would have combined the resource manager of Lippert-Budenske with the central resource manager of Kambatla into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task, as taught by Lippert-Budenske. *Supra* Section VII.C.3.a). The POSITA would have done so in order to gain the additional advantages of Kambatla, namely to enable efficient scheduling of multiple tasks, rather than one computational task, across the processing resources of the heterogeneous computing system, while minimizing "processing duration." EX1006, [0042], [0004], [0017], [0023]-[0024]; EX1003, ¶125.

b) <u>**Element 3.2**</u>

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske-Kambatla includes an *application manager* that receives information from the nodes and processes the information to update an assignment of sub-tasks to the nodes. *Supra* Section VIII.B.1. Lippert-Budenske-Kambatla further includes a *resource manager*. *Supra* Section VIII.B.2.a). Lippert-Budenske includes a resource manager that dynamically alters further distribution of computation nodes and booster nodes during the computing of the computation task. *Supra* Section VII.C.3.b). Further, Kambatla discloses that its central resource manager includes various modules configured to handle different tasks and feed information to other modules "to aid in resource allocation." EX1006, [0036], [0031]-[0035]. Kambatla also discloses that "more or fewer components, organized in various configurations … remain[] within the scope" of Kambatla. *Id.*, [0030]. A POSITA thus would have naturally implemented Lippert-Budenske-Kambatla with a central resource manager as taught by Kambatla that includes a module configured to perform the dynamic-assignment functions of Lippert-Budenske's resource manager, thus achieving the benefits of

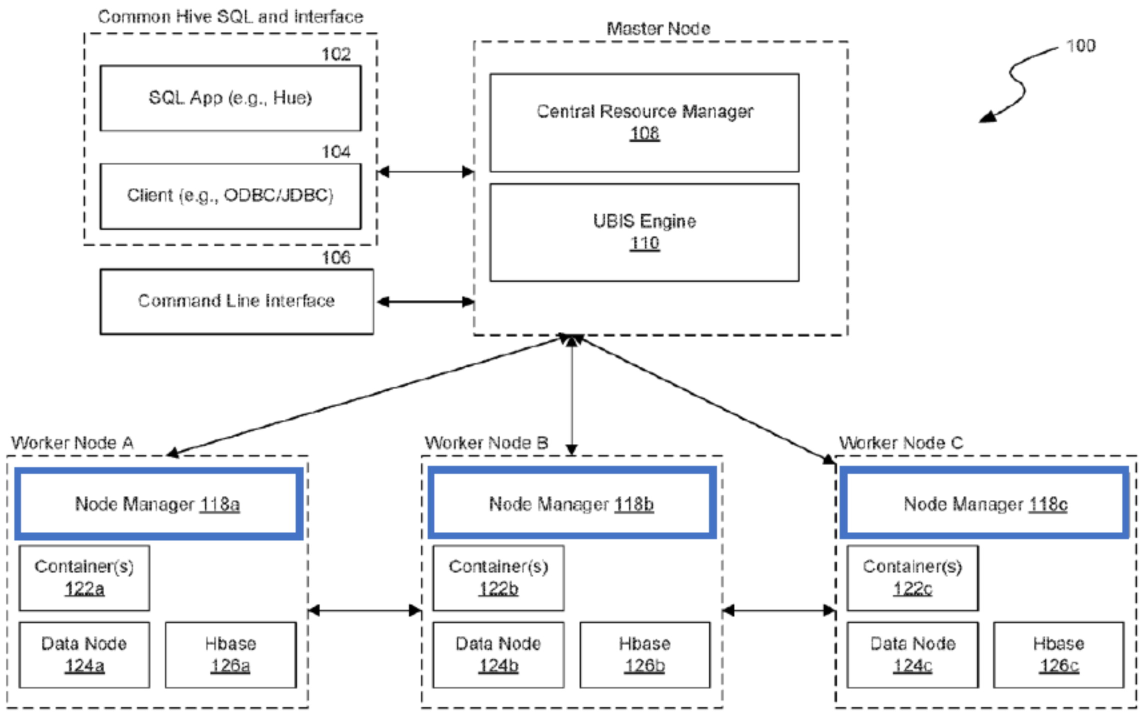
Kambatla's resource manager and the dynamic assignment of Lippert-Budenske's resource manager. *Supra* Sections VII.C.3.b), VIII.B. So implemented, the resource manager of Lippert-Budenske-Kambatla uses the information input by the application manager to dynamically alter further distribution during the computing of the computation task, thus combining the functions of the resource manager of Lippert, the HC Kernel Monitor and Basic Kernel of Budenske, and the resource manager of Kambatla. EX1003, ¶126.

3. <u>Claim 4</u>

Lippert-Budenske-Kambatla teaches and discloses this element, and renders this claim obvious, at least because Lippert-Budenske teaches and discloses a resource manager that receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. *Supra* Sections VIII.B.2, VII.C.4. As Lippert-Budenske-Kambatla includes a resource manager that combines the central resource manager of Kambatla with the resource manager of Lippert-Budenske, the resource manager of Lippert-Budenske-Kambatla receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. EX1003, ¶127.

4. Claim 5

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. Lippert-Budenske renders claim 1 obvious. *Supra* Section VII.C.1. Further, Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.5. Moreover, Kambatla provides further implementation details by expressly disclosing a "resource manager daemon installed at worker nodes in a distributed computing cluster." EX1006, [0037]. The "resource manager daemon installed at the worker nodes includes a node manager 310b, and one or more application masters 320b." *Id.*, [0038], FIGS. 1, 3B. The central resource manager "[c]ommunicat[es] with node managers 118a-c which act as the agents at each node," in order to "allocate and schedule resources available at the various nodes based on the available resources reported from each node manager." *Id.*, [0023].



Id., FIG. 1 (annotated). Each node manager "registers with the [central] resource manager and broadcasts the status of the node including the status of available resources … at the node." *Id.*, [0039]. Further, "[a]pplication masters 320b are responsible for requesting resources from the resource manager 108/110, working with node managers 310b to manage and monitor the containers 330b allocated by the resource manager 108/110." *Id.*, [0041]. Containers are an abstraction of the computing resources of the worker node that are allocated to process a task. *Id.*, [0004], [0023]; EX1003, ¶128.

In combination, Lippert-Budenske-Kambatla includes daemons as taught by Kambatla at each worker node—i.e., at each computation node and each booster node—in order to generate dynamic parameter information as taught by Budenske, as well as information regarding computing capacities and computation task requirements as taught by Lippert. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way because doing so would facilitate efficient communication of the relevant information in a status update for dynamically remapping the sub-tasks among the nodes, while the "additional information [taught by Budenske and Lippert] will typically add little overhead to an existing node [status update]." EX1006, [0039]; EX1003, ¶129.

5. <u>Claim 8</u>

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. *Supra* Section VIII.B.2 (claim 3). Lippert-Budenske teaches and discloses this element. *Supra* Section VII.C.8. Further, according to Kambatla, the daemon operating at a node "monitors the containers 330b at the node … for resource utilization while processes are running" and "calculate[s] the aggregate resource utilization across all containers in the worker node" (*generates a measure of a loading of the node during processing*). EX1006, [0038]-[0040]; EX1003, ¶130.

In combination, Lippert-Budenske-Kambatla includes a daemon operating at each computation node and booster node that generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system as taught by Budenske to also account for "load balancing" as taught by Lippert. EX1004, [0017]; EX1003, ¶131.

6. <u>Claim 9</u>

a) <u>Element [9.1]</u>

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert and Budenske each disclose a heterogeneous computing system. *Supra* Section VII.C.1.a); EX1003, ¶132.

b) <u>Element [9.2]</u>

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. *Supra* Section VII.C.1.a); EX1003, ¶133.

c) <u>Element [9.3]</u>

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2. Kambatla also discloses a "cluster of worker nodes in [] communication (e.g., via computer network) with each other and one or more master nodes." EX1006, [0021]; EX1003, ¶134.

In combination, Lippert-Budenske-Kambatla includes a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes. EX1003, ¶135.

d) **<u>Element [9.4]</u>**

Lippert-Budenske-Kambatla discloses and teaches this element. *Supra* Sections VII.C.1, VIII.B.1, VIII.B.2. EX1003, ¶136.

e) **<u>Element [9.5]</u>**

Lippert-Budenske-Kambatla discloses and teaches this element. *Supra* Sections VII.C.1, VIII.B.1, VIII.B.4. EX1003, ¶137.

7. **<u>Claim 10</u>**

Lippert-Budenske-Kambatla discloses and teaches this claim. *Supra* Sections VIII.B.6, VIII.B.3; EX1003, ¶138.

IX. **<u>NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS</u>**

Petitioner is unaware of any objective indicia of non-obviousness that overcome the above grounds of unpatentability as to any challenged claim. Undersigned Counsel for Petitioner is aware that Patent Owner has made generalized and undeveloped allegations of secondary considerations in a confidential interrogatory response served in district court litigation. Given a protective order in that district court litigation, however, IPR Counsel for Petitioner does not have access to those allegations of secondary considerations and therefore is unable to address them at this stage of the IPR proceedings. Should Patent Owner decide to present any arguments or alleged evidence of secondary considerations in this IPR, Petitioner reserves the right to address such arguments and evidence at the appropriate time. *See, e.g.*, *Microsoft Corp. v. Interdigital Patent Holdings, Inc.*, IPR2024-00305, Paper 12 at 23 (PTAB May 10, 2024).

X. CONCLUSION

Petitioner requests institution for claims 1-10, on Grounds 1 and 2 specified in this petition.

Respectfully submitted,

Dated: December 20, 2024

By: /Andrew M. Mason/

Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITATION PURSUANT TO 37 C.F.R. § 42.24

This brief complies with the type-volume limitation of 37 C.F.R. § 42.24(a)(1)(i).

The brief contains **13,735** words, excluding the parts of the brief exempted by 37 C.F.R. § 42.24(a).

The brief has been prepared in a proportionally spaced typeface using Microsoft Word for O365 in a 14-point Times New Roman font.

Dated: December 20, 2024

By: /Andrew M. Mason/

Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner

CERTIFICATE OF SERVICE IN COMPLIANCE WITH 37 C.F.R. § 42.6(E)(4)

The undersigned certifies that the **PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 11,537,442 and EXHIBITS 1001 – 1025** were served on December 20, 2024, via **Express Mail** on the Patent Owner at the following address of record as listed on the USPTO Patent Center:

Marshall, Gerstein & Borun LLP
233 South Wacker Drive
6300 Willis Tower
Chicago, Il 60606-6357

By: /Andrew M. Mason/

Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner